**Auto Loader:**

Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage without any additional setup.

**How does Auto Loader work?**

Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage. Auto Loader can load data files from AWS S3 (s3://), Azure Data Lake Storage Gen2 (ADLS Gen2, abfss://), Google Cloud Storage (GCS, gs://), Azure Blob Storage (wasbs://), ADLS Gen1 (adl://), and Databricks File System (DBFS, dbfs:/). Auto Loader can ingest JSON, CSV, PARQUET, AVRO, ORC, TEXT, and BINARYFILE file formats.

Auto Loader provides a Structured Streaming source called cloudFiles. Given an input directory path on the cloud file storage, the cloudFiles source automatically processes new files as they arrive, with the option of also processing existing files in that directory. Auto Loader has support for both Python and SQL in Delta Live Tables.

You can use Auto Loader to process billions of files to migrate or backfill a table. Auto Loader scales to support near real-time ingestion of millions of files per hour.

**What is Delta Lake?**

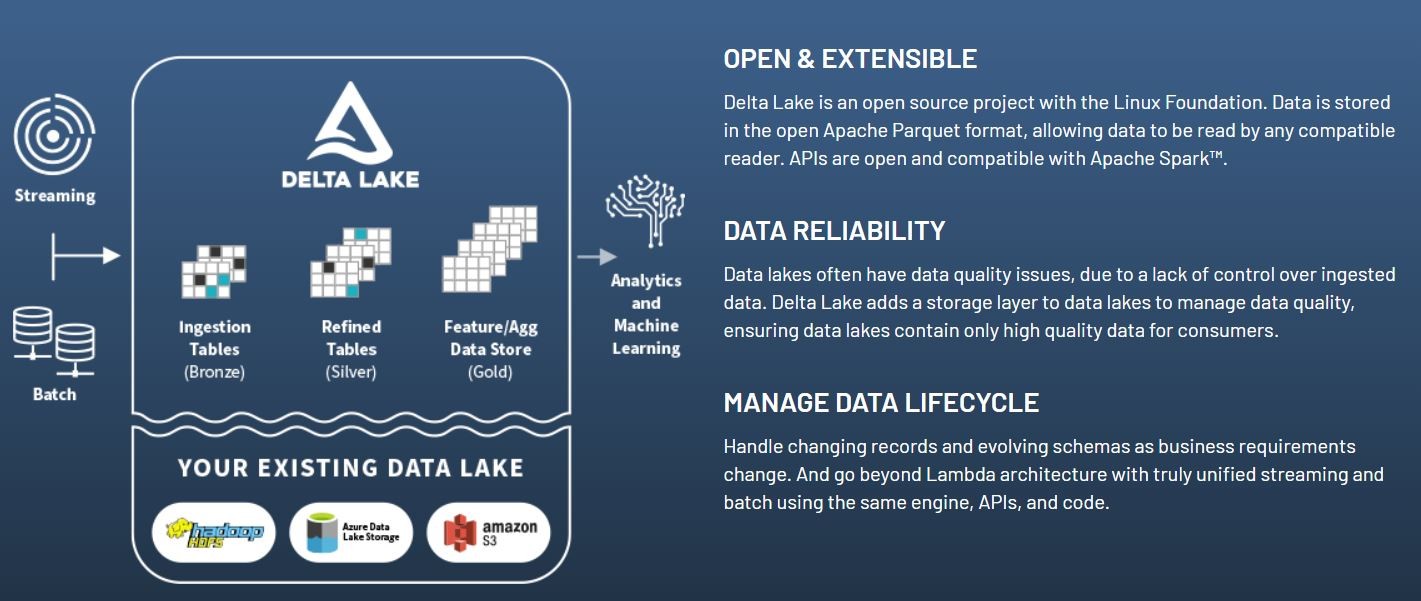
Delta Lake is a technology used for building robust Data Lakes. It a component of Cloud Data Platform. Delta Lake is an open-source storage layer that brings reliability to data lakes. Data Lakes do not support Schema Enforcement, Data Quality, ACID transactions, etc. These drawbacks of Data Lakes can be overcome by use Delta Lake.

**Also, Delta Lake offers the following additional features:**

1. Delta Tables: Provide ACID compliant transactions, Metastore to validate these transactions, Delta Log to keep track of all the transactions since its inception

2. Delta Lake Storage Layer: Stores Delta Tables (files) as well as the Metastore (Metadata) information in Customer's Cloud Storage

3. Delta Architecture: Allows to store Bronze(Raw), Silver(Refined), Gold(Aggregated) versions of data which can be used as Single Source Of Truth for number of applications (Data Science, Analytics, BI).



**What are the advantages of using Delta Tables?**

Following are the advantages of Delta Tables:

1. ACID Compliant. ACID is an acronym for Atomicity (all changes in single operation), Consistency (same data quality before and after operation), Isolation (Independent operation), Durability (Completed operation cannot be undone)

2. Scalable storage as well as scalable metadata

3. Unification of batch & stream processing. Single table can be used as sink/source for batch & stream processing.

4. Time Travel based on automatic versioning and timestamp

**Why Delta Tables are encouraged especially with Databricks and Spark.**

Delta Lake uses Delta Engine for optimizing transactions on Data Lake. Delta Engine is highly compatible with Apache Spark and Spark APIs. And Databricks provides Delta Engine optimizations without additional costs as a built-in feature. So, it is recommended to use Delta Engine optimization features when processing data especially in Databricks or Apache Spark.

**SPARK API:**

The Spark API allows authorized MLS members to request data through developer applications according to the permissions and license requirements of the MLS.

**What Data Is Available Through the API?**

There are three main categories of data available through the API: Listings, Contacts and Market Stats. The access level to each data type is dependent on the role assigned to each user. For example, members of the MLS can access more listing information than consumers on an IDX web site.

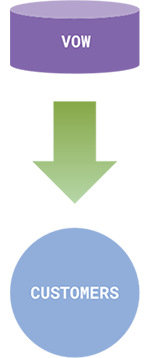
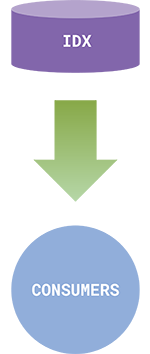
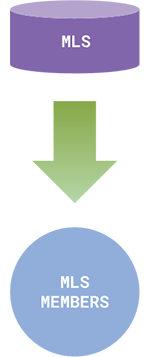
**Role-Based Access to the API**

The API only allows access to data by authorized members of the MLS according to roles set for each user by the MLS through the API Manager in the Platform. In this way, the MLS maintains control of what data each user is able to see through the API and what permission each user has for using the data.

**API Manager Defines Roles for the API**

There can be many different roles for MLS members, but the three most important roles defined by the MLS through the API Role Manager are:

**What is the MLS Member Role?**

The API also allows developers to create applications for MLS members with authority to see the entire MLS database. In other words, using the API, developers can create the same types of applications used by MLS members in the private MLS system.

Example applications:

Listing search and display

CMAs

Prospecting

Email Marketing

Flyers

CRM

Photo Management

Market Analysis

**What are the IDX and VOW Roles?**

The IDX and VOW roles allow authorized MLS members to share listing information with consumers on their web sites or applications and customers who login to the web site or app.

IDX is for Consumers. IDX stands for Internet Data Exchange and is an MLS policy that allows authorized MLS members to display designated MLS information on their web sites and mobile applications to consumers (web site visitors).

Listing search site or app

Open house tours

Single listing sites

Photo Tours

Market Reports

Neighbourhood pages

VOW is for Customers. VOW stands for Virtual Office Website and is an MLS policy that allows authorized MLS members to display designated MLS information on their web sites and mobile applications to their customers. The difference between IDX and VOW primarily is that VOW access requires the consumer to identify themselves and login to the MLS member’s web site or application (their virtual office). Once the consumer is identified and logged in to the MLS member site or app, they can see more information (including sold information) than is available through the IDX role.

**Example applications:**

Customer portals

Home value reports (CMAs)

Hot Sheets

Market analysis reports

Transaction management

**Authentication**

Each request to the API requires two forms of authorization, one from the developer and the other from the MLS member. If either the developer or the MLS member is not authorized to access the API, the request is not allowed. With two forms of authorization, each API request is able to identify the software developer, the developer's application, the end user, and the MLS, and in the case of the VOW role, the agent with whom the end user is working.

End Users -- Private and VOW roles require that the end user, an MLS member or VOW customer, enter a username and password to identify, authenticate, and authorize their active membership in the MLS or their VOW account. The authorization step shows the API that the end user has granted permission for the API requests to occur on that user's behalf. Requests to the API for IDX data require the application to provide an access token that both identifies the developer and the developer's application, and provides proof to the API that the MLS member has personally authorized that application to access data on their behalf and provide that data to IDX visitors. When IDX applications are purchased, the MLS member must authorize that application to access their data.

All forms of authorization used in the API use OAuth 2, a standard that allows authorization to be granted without requiring software developers to store end users' usernames and passwords in their applications, increasing the security of the Platform.

At the time of authentication, users are required to agree to the appropriate Terms of Use.

**Clear and Consistent Licensing and Terms of Use**

Another core focus for the Spark Platform is to ensure that the data and software are licensed under clear and consistent terms of use all the way from the MLS through to agents and their customers.

Terms of use are required to be agreed upon when users license applications through the Store and also when they login to use applications. The authentication process identifies the role or permission level available to that user and instructs the API what data may be provided. The MLS is able to control the data available to each role through the API Manager.

In this way, the license and terms of use for both the MLS data and the software is agreed to consistently by every user gaining access to the data through the API.

**Implementing Data Standards**

One of the core objectives for the Spark Platform is to help MLS organizations implement the Real Estate Standards Organization (RESO) data dictionary being developed.

One of the obstacles historically to implementing data standards was the lack of immediate incentive. Most everyone understands the long-term incentive for more standardized data (lower cost and increased interoperability of software), but those benefits are so long-term that they make action today difficult.

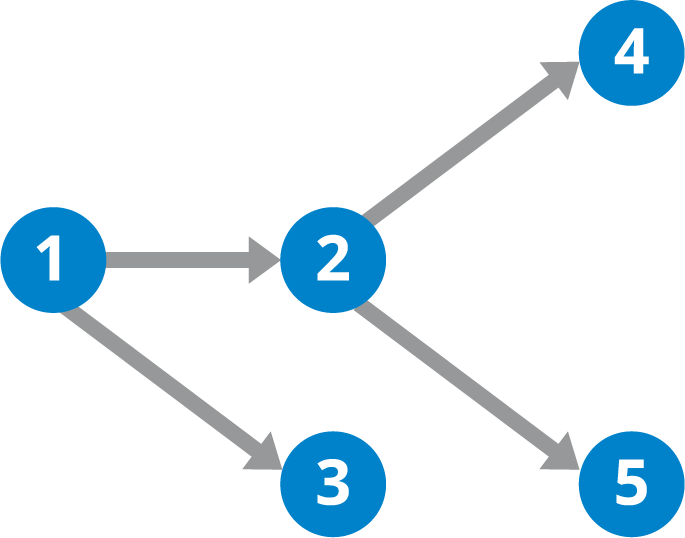
The Spark Platform attempts to address this challenge by creating an economic eco-system that encourages MLSs, brokers and developers to work together to promote more data standards. As FBS brings in data from participating MLSs, the data will be mapped into the RESO standard fields using a Data Field Mapper we’ve created. This software will help both FBS and each MLS work together to continually add more standard fields as they are needed by new applications being created by developers.

Fields not able to be mapped to a standard field currently will still be supported in the API as custom fields. In addition, the API supports localization of the data labels even on standard fields, including the ability to represent the fields in multiple languages. However, all standard fields will be able to be searched through the API by the RESO standard field.

FBS has long been a strong proponent of data standards for the MLS industry and we will continue to work with RESO to expand the data dictionary. As the data dictionary expands, we will be able to map additional fields to the standard fields, with the constant objective of mapping as many fields as possible to the standards.

**Directed Acyclic Graph (DAG)**

A directed acyclic graph (DAG) is a conceptual representation of a series of activities. The order of the activities is depicted by a graph, which is visually presented as a set of circles, each one representing an activity, some of which are connected by lines, which represent the flow from one activity to another. Each circle is known as a “vertex” and each line is known as an “edge.” “Directed” means that each edge has a defined direction, so each edge necessarily represents a single directional flow from one vertex to another. “Acyclic” means that there are no loops (i.e., “cycles”) in the graph, so that for any given vertex, if you follow an edge that connects that vertex to another, there is no path in the graph to get back to that initial vertex.

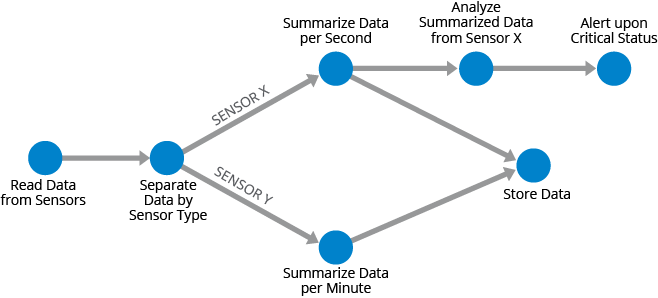


**Directed Acyclic Graph (DAG) Diagram**

**Why Are Directed Acyclic Graphs Useful?**

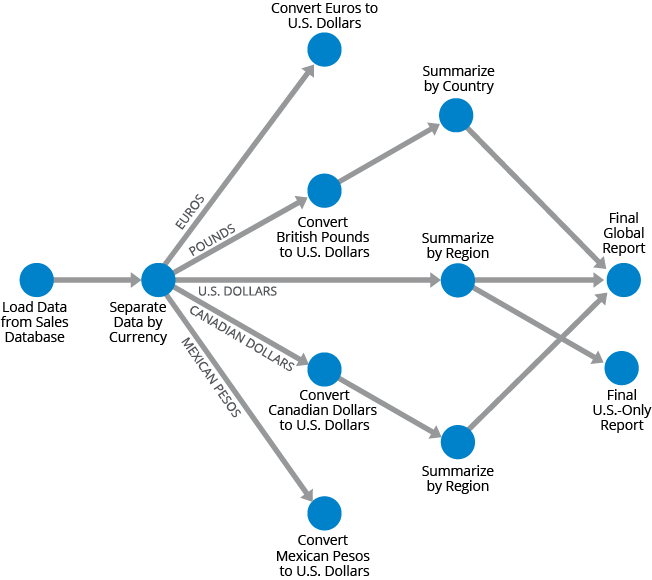
DAGs are useful for representing many different types of flows, including data processing flows. By thinking about large-scale processing flows in terms of DAGs, one can more clearly organize the various steps and the associated order for these jobs. In many data processing environments, a series of computations are run on the data to prepare it for one or more ultimate destinations. This type of data processing flow is often referred to as a data pipeline. As an example, sales transaction data might be processed immediately to prepare it for making real-time recommendations to consumers. As part of the processing lifecycle, the data can go through many steps including cleansing (correcting incorrect/invalid data), aggregation (calculating summaries), enrichment (identifying relationships with other relevant data), and transformation (writing the data into a new format).

One key characteristic of DAGs and the data processing flows that they model is that there can be multiple paths in the flow. This is important because it recognizes the need to process data in multiple ways to accommodate different outputs and needs. In the example flow below, a stream of sensor data is processed. The data is first loaded from the sensors, then are separated by the sensor type. Sensor X data will be summarized per second, and then analyzed in real-time. If any critical status is observed, an alert is sent. The data is also saved for long-term storage and possibly other analysis. Also in this flow is data from sensor Y, which for now is summarized per minute, and then stored in the same long-term store as the data for sensor X.



**A stream of sensor data represented as a directed acyclic graph**

To give an example of how DAGs apply to batch processing pipelines, suppose you have a database of global sales, and you want a report of all sales by region, expressed in U.S. dollars. You might first load all data into a processing engine, separate out data by the different currencies, convert the financial figures to U.S. dollars, summarize the data by country/region, then bring all the data together into a final report. And let’s say that the U.S.-only data will be created into a separate report as well. This data flow could be represented by the DAG shown below.



**Global sales data represented by the directed acyclic graph (DAG).**

Since DAGs apply to both stream and batch processing, it is increasingly common to have hybrid data processing environments that handle both stream and batch data sets. Technologies such as Hazelcast Jet that are designed to handle both types of data let companies build data architectures that take advantage of all of their data.

**Photon runtime**

Photon is the native vectorized query engine on Databricks, written to be directly compatible with Apache Spark APIs so it works with your existing code. It is developed in C++ to take advantage of modern hardware, and uses the latest techniques in vectorized query processing to capitalize on data- and instruction-level parallelism in CPUs, enhancing performance on real-world data and applications-—all natively on your data lake. Photon is part of a high-performance runtime that runs your existing SQL and DataFrame API calls faster and reduces your total cost per workload. Photon is used by default in Databricks SQL warehouses.

**Databricks clusters**

Photon is available for clusters running Databricks Runtime 9.1 LTS and above.

To enable Photon acceleration, select the Use Photon Acceleration checkbox when you create the cluster. If you create the cluster using the clusters API, set runtime\_engine to PHOTON.

Photon supports a number of instance types on the driver and worker nodes. Photon instance types consume DBUs at a different rate than the same instance type running the non-Photon runtime. For more information about Photon instances and DBU consumption, see the Databricks pricing page.

**Photon advantages**

Supports SQL and equivalent DataFrame operations against Delta and Parquet tables.

Accelerates queries that process a significant amount of data (100GB+) and include aggregations and joins.

Faster performance when data is accessed repeatedly from the disk cache.

More robust scan performance on tables with many columns and many small files.

Faster Delta and Parquet writing using UPDATE, DELETE, MERGE INTO, INSERT, and CREATE TABLE AS SELECT, especially for wide tables (hundreds to thousands of columns).

Replaces sort-merge joins with hash-joins.

**Photon coverage**

Operators

Scan, Filter, Project

Hash Aggregate/Join/Shuffle

Nested-Loop Join

Null-Aware Anti Join

Union, Expand, ScalarSubquery

Delta/Parquet Write Sink

Sort

Window Function

**Expressions**

* Comparison / Logic
* Arithmetic / Math (most)
* Conditional (IF, CASE, etc.)
* String (common ones)
* Casts
* Aggregates(most common ones)
* Date/Timestamp

**Data types**

* Byte/Short/Int/Long
* Boolean
* String/Binary
* Decimal
* Float/Double
* Date/Timestamp
* Struct
* Array
* Map

The following table lists supported Databricks expressions and the minimum Databricks Runtime release version that supports it.

| **Name** | **Release** |
| --- | --- |
| Abs | Databricks Runtime 8.3 |
| Acos | Databricks Runtime 10.4 LTS |
| Add | Databricks Runtime 8.3 |
| AddMonths | Databricks Runtime 8.3 |
| AesDecrypt | Databricks Runtime 10.4 LTS |
| AesEncrypt | Databricks Runtime 10.4 LTS |
| And | Databricks Runtime 8.3 |
| ArrayContains | Databricks Runtime 8.3 |
| ArrayDistinct | Databricks Runtime 10.0 |
| ArrayExcept | Databricks Runtime 10.1 |
| ArrayExists | Databricks Runtime 10.4 LTS |
| ArrayFilter | Databricks Runtime 10.4 LTS |
| ArrayForAll | Databricks Runtime 10.4 LTS |
| ArrayIntersect | Databricks Runtime 10.1 |
| ArrayJoin | Databricks Runtime 10.4 LTS |
| ArraySize | Databricks Runtime 10.4 LTS |
| ArrayTransform | Databricks Runtime 10.4 LTS |
| ArrayUnion | Databricks Runtime 10.1 |
| Atan | Databricks Runtime 9.1 LTS |
| Atan2 | Databricks Runtime 9.1 LTS |
| Average | Databricks Runtime 8.3 |
| Base64 | Databricks Runtime 9.1 LTS |
| Bin | Databricks Runtime 10.0 |
| BitAndAgg | Databricks Runtime 8.3 |
| BitLength | Databricks Runtime 11.3 LTS |
| BitOrAgg | Databricks Runtime 8.3 |
| BitwiseAnd | Databricks Runtime 8.3 |
| BitwiseNot | Databricks Runtime 8.3 |
| BitwiseOr | Databricks Runtime 8.3 |
| BitwiseReverse | Databricks Runtime 8.3 |
| BitwiseXor | Databricks Runtime 8.3 |
| BitXorAgg | Databricks Runtime 8.3 |
| BoundaryAsGeojson | Databricks Runtime 11.3 LTS |
| BoundaryAsWkb | Databricks Runtime 11.3 LTS |
| BoundaryAsWkt | Databricks Runtime 11.3 LTS |
| Cast | Databricks Runtime 8.3 |
| Cbrt | Databricks Runtime 8.4 |
| CeilExpressionBuilder | Databricks Runtime 8.3 |
| CenterAsGeojson | Databricks Runtime 11.3 LTS |
| CenterAsWkb | Databricks Runtime 11.3 LTS |
| CenterAsWkt | Databricks Runtime 11.3 LTS |
| Chr | Databricks Runtime 10.1 |
| Coalesce | Databricks Runtime 8.3 |
| CollectList | Databricks Runtime 9.0 |
| Concat | Databricks Runtime 8.3 |
| ConcatWs | Databricks Runtime 8.3 |
| Conv | Databricks Runtime 8.3 |
| Cos | Databricks Runtime 10.4 LTS |
| Count | Databricks Runtime 8.3 |
| CreateArray | Databricks Runtime 8.3 |
| CreateMap | Databricks Runtime 8.4 |
| CreateNamedStruct | Databricks Runtime 8.3 |
| CreateStruct | Databricks Runtime 8.3 |
| CurrentCatalog | Databricks Runtime 8.3 |
| CurrentDatabase | Databricks Runtime 8.3 |
| CurrentDate | Databricks Runtime 8.3 |
| CurrentTimestamp | Databricks Runtime 8.3 |
| CurrentTimeZone | Databricks Runtime 8.3 |
| CurrentUser | Databricks Runtime 8.3 |
| DateAdd | Databricks Runtime 8.3 |
| DateDiff | Databricks Runtime 8.3 |
| DateFormatClass | Databricks Runtime 8.3 |
| DateFromUnixDate | Databricks Runtime 8.3 |
| DateSub | Databricks Runtime 8.3 |
| DayOfMonth | Databricks Runtime 8.3 |
| DayOfWeek | Databricks Runtime 8.3 |
| DayOfYear | Databricks Runtime 8.3 |
| Decode | Databricks Runtime 8.3 |
| DenseRank | Databricks Runtime 10.4 LTS |
| Divide | Databricks Runtime 8.3 |
| ElementAt | Databricks Runtime 8.3 |
| EqualNullSafe | Databricks Runtime 8.3 |
| EqualTo | Databricks Runtime 8.3 |
| Exp | Databricks Runtime 8.4 |
| Explode | Databricks Runtime 8.4 |
| Extract | Databricks Runtime 8.3 |
| First | Databricks Runtime 8.3 |
| FloorExpressionBuilder | Databricks Runtime 8.3 |
| FromUnixTime | Databricks Runtime 8.3 |
| FromUTCTimestamp | Databricks Runtime 8.3 |
| Get | Databricks Runtime 11.3 LTS |
| GetJsonObject | Databricks Runtime 11.2 |
| GreaterThan | Databricks Runtime 8.3 |
| GreaterThanOrEqual | Databricks Runtime 8.3 |
| Greatest | Databricks Runtime 8.3 |
| GridDistance | Databricks Runtime 11.3 LTS |
| H3ToString | Databricks Runtime 11.3 LTS |
| Hex | Databricks Runtime 9.1 LTS |
| Hour | Databricks Runtime 8.3 |
| If | Databricks Runtime 8.3 |
| In | Databricks Runtime 8.3 |
| InitCap | Databricks Runtime 11.3 LTS |
| InputFileBlockLength | Databricks Runtime 8.3 |
| InputFileBlockStart | Databricks Runtime 8.3 |
| InputFileName | Databricks Runtime 8.3 |
| InSet | Databricks Runtime 8.3 |
| IntegralDivide | Databricks Runtime 8.3 |
| IsChildOf | Databricks Runtime 11.3 LTS |
| IsNaN | Databricks Runtime 8.3 |
| IsNotNull | Databricks Runtime 8.3 |
| IsNull | Databricks Runtime 8.3 |
| IsPentagon | Databricks Runtime 11.3 LTS |
| IsValid | Databricks Runtime 11.3 LTS |
| JsonToStructs | Databricks Runtime 11.2 |
| Lag | Databricks Runtime 10.4 LTS |
| Last | Databricks Runtime 10.4 LTS |
| LastDay | Databricks Runtime 8.3 |
| Lead | Databricks Runtime 10.4 LTS |
| Least | Databricks Runtime 8.3 |
| Length | Databricks Runtime 8.3 |
| LengthOfJsonArray | Databricks Runtime 11.1 |
| LessThan | Databricks Runtime 8.3 |
| Levenshtein | Databricks Runtime 10.1 |
| Like | Databricks Runtime 8.3 |
| Log | Databricks Runtime 8.3 |
| Log2 | Databricks Runtime 8.4 |
| LongLatAsH3 | Databricks Runtime 11.3 LTS |
| LongLatAsH3String | Databricks Runtime 11.3 LTS |
| Lower | Databricks Runtime 8.3 |
| LPadExpressionBuilder | Databricks Runtime 8.3 |
| MakeDate | Databricks Runtime 8.3 |
| MakeTimestamp | Databricks Runtime 8.3 |
| Max | Databricks Runtime 8.3 |
| MaxChild | Databricks Runtime 11.3 LTS |
| Md5 | Databricks Runtime 10.4 LTS |
| MicrosToTimestamp | Databricks Runtime 8.3 |
| MillisToTimestamp | Databricks Runtime 8.3 |
| Min | Databricks Runtime 8.3 |
| MinChild | Databricks Runtime 11.3 LTS |
| Minute | Databricks Runtime 8.3 |
| MonotonicallyIncreasingID | Databricks Runtime 8.3 |
| Month | Databricks Runtime 8.3 |
| MonthsBetween | Databricks Runtime 8.3 |
| Multiply | Databricks Runtime 8.3 |
| Murmur3Hash | Databricks Runtime 8.3 |
| NaNvl | Databricks Runtime 8.3 |
| NextDay | Databricks Runtime 8.3 |
| Not | Databricks Runtime 8.3 |
| Now | Databricks Runtime 8.3 |
| NthValue | Databricks Runtime 10.4 LTS |
| NTile | Databricks Runtime 10.4 LTS |
| NullIf | Databricks Runtime 8.3 |
| Nvl | Databricks Runtime 8.3 |
| Nvl2 | Databricks Runtime 8.3 |
| OctetLength | Databricks Runtime 8.3 |
| ParseToDate | Databricks Runtime 8.3 |
| ParseToTimestamp | Databricks Runtime 8.3 |
| Percentile | Databricks Runtime 10.4 LTS |
| PercentRank | Databricks Runtime 10.4 LTS |
| Pi | Databricks Runtime 8.3 |
| Pmod | Databricks Runtime 8.3 |
| PosExplode | Databricks Runtime 9.1 LTS |
| Pow | Databricks Runtime 8.3 |
| Quarter | Databricks Runtime 8.3 |
| Rand | Databricks Runtime 8.3 |
| Rank | Databricks Runtime 10.4 LTS |
| RegExpExtract | Databricks Runtime 8.3 |
| RegExpExtractAll | Databricks Runtime 11.1 |
| RegExpReplace | Databricks Runtime 9.1 LTS |
| RegrAvgX | Databricks Runtime 10.5 |
| RegrAvgY | Databricks Runtime 10.5 |
| Remainder | Databricks Runtime 8.3 |
| Resolution | Databricks Runtime 11.3 LTS |
| Reverse | Databricks Runtime 8.3 |
| Reverse | Databricks Runtime 8.3 |
| RLike | Databricks Runtime 8.3 |
| Round | Databricks Runtime 8.3 |
| RowNumber | Databricks Runtime 10.4 LTS |
| RPadExpressionBuilder | Databricks Runtime 8.3 |
| Second | Databricks Runtime 8.3 |
| SecondsToTimestamp | Databricks Runtime 8.3 |
| Sha1 | Databricks Runtime 10.4 LTS |
| Sha2 | Databricks Runtime 10.4 LTS |
| ShiftLeft | Databricks Runtime 8.3 |
| ShiftRight | Databricks Runtime 8.3 |
| ShiftRightUnsigned | Databricks Runtime 8.3 |
| Sin | Databricks Runtime 10.4 LTS |
| Size | Databricks Runtime 8.3 |
| Slice | Databricks Runtime 8.3 |
| SoundEx | Databricks Runtime 10.1 |
| SparkVersion | Databricks Runtime 8.3 |
| Sqrt | Databricks Runtime 8.4 |
| StddevPop | Databricks Runtime 8.3 |
| StddevSamp | Databricks Runtime 8.3 |
| StringInstr | Databricks Runtime 8.3 |
| StringLocate | Databricks Runtime 8.3 |
| StringRepeat | Databricks Runtime 11.2 |
| StringSpace | Databricks Runtime 8.3 |
| StringSplit | Databricks Runtime 8.3 |
| StringToH3 | Databricks Runtime 11.3 LTS |
| StringTranslate | Databricks Runtime 10.4 LTS |
| StringTrim | Databricks Runtime 8.3 |
| StringTrimBoth | Databricks Runtime 8.3 |
| StringTrimLeft | Databricks Runtime 8.3 |
| StringTrimRight | Databricks Runtime 8.3 |
| StructsToJson | Databricks Runtime 11.1 |
| Substring | Databricks Runtime 8.3 |
| Subtract | Databricks Runtime 8.3 |
| Sum | Databricks Runtime 8.3 |
| Tan | Databricks Runtime 9.1 LTS |
| ToChildren | Databricks Runtime 11.3 LTS |
| ToParent | Databricks Runtime 11.3 LTS |
| ToRadians | Databricks Runtime 10.1 |
| ToUnixTimestamp | Databricks Runtime 8.3 |
| ToUTCTimestamp | Databricks Runtime 8.3 |
| TruncDate | Databricks Runtime 8.3 |
| TruncTimestamp | Databricks Runtime 8.3 |
| TryElementAt | Databricks Runtime 10.0 |
| TryValidate | Databricks Runtime 11.3 LTS |
| UnaryMinus | Databricks Runtime 8.3 |
| UnBase64 | Databricks Runtime 9.1 LTS |
| Unhex | Databricks Runtime 9.1 LTS |
| UnixDate | Databricks Runtime 8.3 |
| UnixMicros | Databricks Runtime 8.3 |
| UnixMillis | Databricks Runtime 8.3 |
| UnixSeconds | Databricks Runtime 8.3 |
| UnixTimestamp | Databricks Runtime 8.3 |
| Upper | Databricks Runtime 8.3 |
| Uuid | Databricks Runtime 8.3 |
| Validate | Databricks Runtime 11.3 LTS |
| VarianceSamp | Databricks Runtime 10.1 |
| WeekDay | Databricks Runtime 8.3 |
| WeekOfYear | Databricks Runtime 8.3 |
| XxHash64 | Databricks Runtime 10.0 |
| Year | Databricks Runtime 8.3 |

Limitations

* Structured Streaming: Photon currently supports stateless streaming with Delta, Parquet, and CSV. Kafka and Kinesis support is in [Public Preview](https://docs.databricks.com/release-notes/release-types.html)
* Does not support UDFs.
* Does not support RDD APIs.
* Not expected to improve short-running queries (<2 seconds), for example, queries against small amounts of data.

Features not supported by Photon run the same way they would with Databricks Runtime; there is no performance advantage for those features.

**External vs Internal Tables:**

An internal table data is stored in the warehouse folder, whereas an external table data is stored at the location you mentioned in table creation. So when you delete an internal table, it deletes the schema as well as the data under the warehouse folder, but for an external table it's only the schema that you will loose.

**Difference between Internal & External tables**

**For External Tables:**

* External table stores files on the HDFS server but tables are not linked to the source file completely.
* If you delete an external table the file still remains on the HDFS server.
* As an example, if you create an external table called “table\_test” in HIVE using HIVE-QL and link the table to file “file”, then deleting “table\_test” from HIVE will not delete “file” from HDFS.
* External table files are accessible to anyone who has access to HDFS file structure and therefore security needs to be managed at the HDFS file/folder level.
* Meta data is maintained on master node, and deleting an external table from HIVE only deletes the metadata not the data/file.

**For Internal Tables-**

* Stored in a directory based on settings in hive.metastore.warehouse.dir, by default internal tables are stored in the following directory “/user/hive/warehouse” you can change it by updating the location in the config file.
* Deleting the table deletes the metadata and data from master-node and HDFS respectively.
* Internal table file security is controlled solely via HIVE. Security needs to be managed within HIVE, probably at the schema level (depends on organization).
* Hive may have internal or external tables, this is a choice that affects how data is loaded, controlled, and managed.

**Use EXTERNAL tables when:**

* The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn’t lock the files.
* Data needs to remain in the underlying location even after a DROP TABLE. This can apply if you are pointing multiple schema (tables or views) at a single data set or if you are iterating through various possible schema.
* Hive should not own data and control settings, directories, etc., you may have another program or process that will do those things.
* You are not creating table based on existing table (AS SELECT).

**Use INTERNAL tables when:**

* The data is temporary.
* You want Hive to completely manage the life-cycle of the table and data.